Exploring Embedded Symmetric Multiprocessing with Various On-Chip Architectures

Gorker Alp Malazgirt*, Bora Kiyan†, Deniz Candas‡, Kamil Erdayandi* and Arda Yurdakul*
*Department of Computer Engineering
Bogazici University, Istanbul, Turkey
Email: alp.malazgirt, kamil.erdayandi, yurdakul@boun.edu.tr
†Robert College, Istanbul, Turkey
Email: bora.kiyan@gmail.com
‡Istanbuler Gymnasium, Istanbul, Turkey
Email: dnzcandas@gmail.com

***Abstract*—Multicore embedded systems have evolved to appear in different domains. In this paper, we explore and compare various on-chip architectures with respect to a number design metrics. Unlike earlier published works that majorly concern with optimizations in processor, memory and cache hierarchies, in this paper, we aim to ascertain the best on-chip architectures for given processor cores, Level 1-2-3 caches modeled from Intel Atom embedded processor family. We investigate topologies that haven't been considered before for symmetric multiprocessing in embedded systems domain. These architectures consist of shared instruction caches between cores and heterogenous cache topologies that feature bypassing a level in the cache hierarchy. Through our experiments with multithreaded workloads, we elicit the unconventional topologies that could provide more performance and energy efficiency than regular topologies. In addition, using our experimental data, we conclude that certain design metrics could depend on given workload, however there also exists some metrics that are more dependent on the underlying topologies. Thus, we urge the need for future exploration tools to gather the necessary metrics while choosing the appropriate SMP architectures.**

I. INTRODUCTION

In the last decade, embedded systems have evolved into new concepts like Cyber Physical Systems and Internet-of-Things. These systems have to be designed for specific functions under extreme design constraints such as low-power, timely response with an admissable quality of service, lightweight, low-cost etc. High-end systems like smart phones tend to support more applications from different domains such as multimedia streaming and gaming which require high performance. In contrast, low-end systems like sensory devices try to provide more ubiquitous experiences with extended battery lifetimes. As a result, different types of processing platforms are available on the embedded market.

While designing an embedded system, the processing platform is selected according to the benchmarks or previous design experiences. In this design paradigm, the architecture of the processing platform is initially fixed. Design space exploration tools [1] find the best memory size for the "given" architecture. These tools operate jointly with powerful cache [2], memory [3], architectural [4] and energy [5] simulators so as to evaluate the energy or performance of a point in the design evaluation space and iterates between an optimizer and a simulator until sufficiently good result is achieved. Design space exploration tools make use of not only cycle-accurate simulators that support different levels of hardware customizations [6], but also compilers [7] that allow numerous optimizations for different processor architectures from given applications, as well. These tools are sometimes accepted as inadequate and hidebound for providing successful design of embedded systems; especially the architectural design space exploration provided by cycle accurate simulators are overwhelmingly large but very beneficial.

Embedded processors are used extensively in areas ranging from embedded systems to high performance computing [8]. In these different domains, the processors are configured with different configurations. For instance, when an Intel Atom processor is configured for a laptop, L3 cache is included, whereas for a handheld device, L3 cache is omitted due to power requirements [9]. When we investigated the processor product lineups, we saw that *the number of cores and clock speed*, *the number of instruction/data caches* and *their connectivity* are three most important design parameters that differ [10]. Hence, we are interested in exploring the design space that is depended on the aforementioned parameters.

This work's contributions can be summarized below:

- Exploring SMP with unconventional on-chip architectures such as shared L1 instruction caches, heterogenous exposure of L1 and L2 caches from processor cores to main memory and combining bus interconnection with ring topology in order to support aforementioned heterogeneity
- Analyzing and identifying workload dependent and architecture dependent design metrics which help in selecting on-chip architectures from simulation outputs

We have experimented with four different benchmarks. We have shown that unconventional heterogenous cache connections can provide performance and power advantages over conventional hierarchies. The major differences of our work to other works are that available works have not investigated SMP with unconventional architectures in the embedded domain. In addition, we have analyzed our experimental results, and elaborated the necessity of using multiple metrics for architecture exploration.

978-1-4673-8299-1/15 $31.00 © 2015 IEEE
DOI 10.1109/EUC.2015.19

The number of variables and workloads in this work might not cover the broad embedded systems design space. However, we have explored and constructed the design space with parameters which have not been studied broadly in embedded systems and SMP domain before. Additionally, in this study we have elevated the importance of designing with numerous metrics instead of a single metric.

In the next section, we present the related works in the literature. Our reference architecture is explained in Section III, and Section IV discusses the generation of on-chip architectures. Section V presents our results. We finalize our paper with our discussion and our conclusion in Section VI.

II. Related Work

There has been a considerable amount of research work aiming to explore the best architecture, the global optimization of such large design space is inherently a hard problem [11]. Our work differentiates from other works in three different ways. First of all, our work explores the design space of embedded architectures at the component level, such as number of cores, number and types of caches and on chip connectivity. Second, we explore sharing L1 instruction caches in a general purpose computing environment. Lastly, we explore heterogenous connection of processor cores to main memory via different levels and types of caches. In addition, we discuss the necessity of architecture exploration by using multiple design metrics instead of a single metric. According to our investigations, there has not been any work that presents in the same context as this work.

Our work differs from design space exploration tools which focus on cache hierarchy based methods because our tool does not consider the modification of cache architectures [12], [13] or employ configurable caches [14]. Instead, our L1, L2 and L3 cache architectures are fixed, thus we explore the connectivity and instances of these caches with processor cores and main memory. Thus, we explore unconventional topologies which has not shown previously such as asymmetrical L3 exposure to processor cores. However, our methodology can be combined with methods that explore cache structures. Nevertheless, this addition of cache structure optimization would increase the design space.

The work in [15] explores different on-chip memory hierarchies based on energy cost models and aims to minimize resources. In our work, we elaborate the importance of designing with multiple metrics instead of only energy. We do not take into account the memory space partitioning problem for the intercommunication of processor cores. We use shared memory communication models which is supported by programming APIs such as OpenMP.

Authors in [12] apply designtime/runtime combined approach for creating power efficient architectures for embedded systems. They identify hardware and software parameters for design space exploration for minimum power and execution time. However, since different applications require different hardware requirements, they choose to average over different applications for the best fit. In our work, we define scenarios by combining different applications which are likely to occur in real life cases.

Shared instruction caches have been researched in SIMD based architectures since it provides more energy efficiency than multiple instruction caches. The presented work in [16] has explored the benefits of instruction caches in general purpose computing. The authors have shown that in several signal processing applications, shared caches with a special multicast data distribution feature fits in smaller area and provides a similar performance with respect to a conventional private cache design. In this work, we do not explore the structure of the instruction cache. Instead we explore sharing L1 instruction caches in a general purpose computing environment.

The authors in [17] have explored parameters of shared instruction cache architectures in multiprocessors. For shared instruction caches, they have built logarithmic interconnects and divided the cache to different banks. In this way, they have aimed to reduce instruction fetch contentions. Our work do not customize the interconnect structure or the structure of the caches such as the number of cache memory banks. We try different connection topologies of caches and cores. The authors of [17] have increased performance with shared instruction caches when the number of banks and the amount parallelism from the given application match. However, we increase performance through exploring different types of on-chip topologies.

III. Reference Multicore Architecture

Our reference multicore architecture is a shared memory symmetric multi processing system (SMP) and each processor core supports up to three levels of cache memory. The communication between cores and memory units are handled by bus or ring interconnection networks. There are various possibilities to design a shared memory SMP, however it is not feasible to investigate all topologies, because generated topologies have to be realistic and suitable for embedded systems domain.

Figure 1 presents an example topology from our topology pool. This topology has four cores and four Level 1 (L1) data caches. Two L1 instruction caches are shared by the four cores. The caches are connected to higher levels via a bus interconnection network. At the second level, there are two Level 2 (L2) data caches and a single L2 instruction cache. Similarly, L2 caches are connected to L3 caches via bus interconnection. At the last level, there is a ring topology which connects L3 data cache, L2 instruction cache and the memory. There exists a memory management unit that handles the data connection between the nodes in the ring topology and the main memory which is an off-chip hardware block. However, this unit is not shown. We call the example topology shown in Figure 1 a "bypass" topology. In general, we divide our topologies into three categories:

Regular: Regular topologies consist of symmetric connections of core to memory. If all cores are Harvard architectures, their caches are private. They don't share instruction caches with any other core. In addition, in regular topologies, cache sharing between cores occur only if cores are Von Neumann architectures because caches provide both data and instruction to all cores. Regular topologies are used extensively today in all commercial embedded processors [10], [18].

Hybrid: Hybrid constructions allow sharing caches between cores in Harvard architectures. For instance in a four core topology, each pair of cores can share an L1 instruction cache. Similarly, in a 2-core system, both cores can share an instruction cache at the L1 and L2 level. In our experiments, we have observed that architectures that have shared data caches and private instruction caches have caused significant performance degradation, therefore shared data caches are not considered.

By-pass: By-pass topology is the denotation for any kind of topology that features a level jump in the hierarchy. For instance an L1 cache may be connected to an L3 cache, bypassing the L2 level. An L2 cache may be connected to Main Memory, bypassing L3 level; and if there is no L3, an L1 cache may be connected to the Main Memory, bypassing the L2 level. This unconventional type of topology architecture prevents the redundant usage of caches in cases where limited instructions or data is utilized. This distinctive topology prevents the overcrowding of CPU space as well as speeding up the CPU.

The processor cores for each topology are identical. At each level, we have determined the structures of the caches. The bandwidth of the interconnection networks increase towards higher levels, thus resembles a fat tree network. The process of generating topologies are explained in Section IV and the specifications of each hardware block used in the experiments are explained in Section V.

IV. Topology Generation

In this section, we introduce our on-chip topologies. We generate different topologies based on the rules that we have identified. These rules yield systematically generated topologies.

- **Rule 1**: Each core can either have a shared instruction and data cache or they can be separate and private to the core. Namely, each core can have either a Von Neumann or a Harvard topology. This is shown in Figure 2.
- **Rule 2**: In Harvard topologies, two cores can share an instruction cache. Figure 3 shows our two-core template. The four- core version is obtained by mirroring the template. In Von Neumann architectures, two and four cores can be connected to a single cache. In addition, when instruction and data caches are private at a lower level, they never split when going higher levels. However, a data cache and an instruction cache can be connected to a cache at a higher level.
- **Rule 3**: When building hybrid topologies, we have always chosen the number of instruction caches equal to or less than the number of data caches. Our experiments have shown that the topologies which have a higher number of instruction caches than data caches have not provided any performance or energy efficiency advantage.
- **Rule 4**: The number of cache levels in each topology is either two or three. There does not exist any topology with only L1 caches, because it would not be possible to explore the effects of hybrid or bypass topologies when there is only a single level of caches.
- **Rule 5**: Let α_x denote the number of caches at level x, then $\alpha_3 < \alpha_2 \leq \alpha_1$. However, this rule is violated when the number of L2 caches equal one and an L3 cache is generated. Nevertheless, this rule is satisfied in all other topologies.
- **Rule 6**: The topologies don't include separate data or instruction caches in Level 3 due to their higher costs in terms of area.
- **Rule 7**: Only the instruction caches are bypassed, because experiments have showed that more data streaming is required by software compared to the instruction fetching provided to the processor cores.

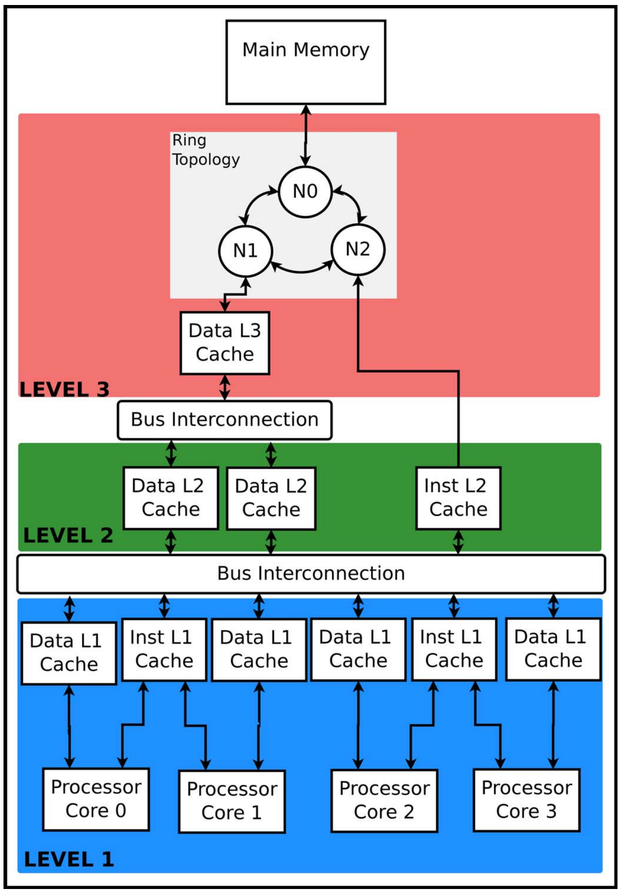

Fig. 1: An example topology which is shown introduce our different SMP architectures

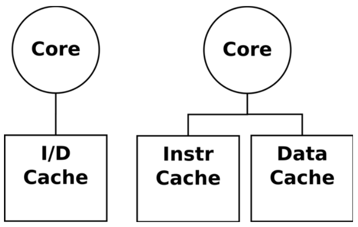

Fig. 2: A processor core can either have shared or private instruction and data cache (Rule 1)

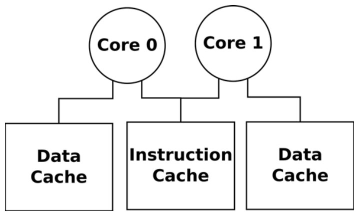

Fig. 3: The template which is used to create topologies that share instruction caches (Rule 2)

Parameter	Number	Type
Core	2, 4	Von Neumann/Harvard
L1 Caches	1, 2, 4, 8	Seperate/Shared Instrution and Data
L2 Caches	1, 2, 4, 8	Seperate/Shared Instruction and Data
L3 Caches	0, 1, 2, 4	Shared Instruction and Data

TABLE I: Design space for the target SMP topologies

Thus, data caches are not bypassed.

The target design space is composed of the parameters related to the types of cores, L1, L2, L3 caches and their connectivity. A detailed view of the parameters and their values is shown in Table I. In particular, the type of a core determines if the processor has dedicated instruction and data caches (Harvard) or not (Von Neumann). Similarly, L1 and L2 caches can be instruction and data caches separately, however at L3, they are always shared. There are a couple exceptions that an instruction cache is bypassed at Level 3, thus only the data cache at Level 2 is connected to the L3 cache.

Based on the rules, 133 topologies could be generated. Among these 133 topologies, we have selected 67 of them for experiments. We have opted to neglect hybrid topologies that have more than one L3 cache because our observations have shown that multiple L3 caches have diminished the performance and energy efficiency of shared instruction caches. We have omitted bypass topologies with more than two L3 caches, because the ring topology required at the last level has increased in size and latency. This has diminished the advantage of having a bypass topology.

V. Experiments

A. The Simulation Environment and Workload preparation

In this section, we present the results of our experiments. The results are obtained by simulating topologies explained in the previous section. The topologies are experimented with four different workloads.

We have taken three benchmarks from MiBench [19] and PARSEC [20] benchmark suites. We have combined the benchmarks and created real life cases. Benchmarks consist of Blackscholes [20], x264 [20] and Dijsktra [19]. Black-Scholes is a partial differential equation based algorithm and it is used in stocks trading. The x264 application is an H.264/AVC (Advanced Video Coding) video encoder. The Dijkstra benchmark constructs a large graph in an adjacency matrix representation and then calculates the shortest path between every pair of nodes using repeated applications of Dijkstras algorithm. It is used in networking applications for DNS Look-up and IP searching.

1) Case 1: x264 + Network - Streaming video scenario in TVs or consoles
2) Case 2: Blackscholes + Network- Stocks trading from a handheld divide
3) Case 3: Blackscholes + Network + x264 - Multitasking previous two scenarios in a smart phone
4) Case 4: Network - A wireless sensor node which is communicating with a number of nodes

Evaluating the results of topologies from a single benchmark might be inadequate because the topology could be favored by the given workload. However, in a real life scenario, workloads tend to occur in combinations of benchmarks that are similar to the above cases. Therefore, it is best to consider combinations of these distinct benchmarks. The real life cases are formed by calculating the median values of simulation results of each workload. Thus, the workloads contribute 50% to the execution of each case. Furthermore, the ratio of workload contributions can be changed. This technique is also applied by the EEMBC benchmarks [21]. To simulate and measure the performance of each topology for the four cases, Multi2Sim [4] is utilized. In this work, we use Multi2Sim in full system simulation mode with dynamic context switching enabled for multithreaded workloads. The cache coherency is handled by Multi2Sim by implementing NMOESI protocol which is an extension to MOESI protocol [22]. In our experiments, we have not applied any power saving methods such as voltage scaling or frequency lowering in the simulator. All the workloads are compiled with GCC 4.9 and three compiler directives are used. These are processor architecture definition, functional inlining and static compilation. We have observed that processor architecture definition and functional inlining significantly increased simulation runtime performance. Static compiler directive has increased the simulation time. Each case is executed with the number of threads that is equal to the number of cores in the topology under test.

The details of processor cores, caches and memory hardware that we have used in are experiments, are shown in Table II. We have used Intel Architecture Descriptions [10] while modeling the hardware in Multi2Sim. The processor core is modeled to resemble Intel Atom 32 nm Saltwell architecture and it is clocked at 1 GHz. The Intel Atom Family establishes a substantial compatibility for embedded system; proving to be essential processor for our purposes. In order to comply with Atom processor cores, the cache configurations are excerpted from the Atom product line topologies. As a whole, the system model is created analogous to a real Atom based computer because consistency among the processor caches and memory is complemented accordingly.

The by-pass topologies explained in Section III requires a ring topology which is built at the last cache level. The nodes of the ring topology are connected to the caches and the main memory. Thus, the memory management unit handles the data transfer between the caches and the memory. The data handling mechanism does not require any additional mechanism to implement in the workloads. Multi2Sim handles the memory management between the cache memories and the

Components	Configurations
Core	2 or 4 Atom cores with 1 thread in each core
L1 Cache	Sets = 64
	Assoc = 8
	BlockSize = 64
	Policy = LRU
	Latency = 5 cycles
	Ports = 2
L2 Cache	Sets = 512
	Assoc = 8
	BlockSize = 64
	Policy = LRU
	Latency = 12 cycles
	Ports = 2
L3 Cache	Sets = 8192
	Assoc = 16
	BlockSize = 64
	Policy = LRU
	Latency = 36 cycles
	Ports = 2
Memory	BlockSize = 256
	Latency = 93 cycles
	Ports = 2
Network	DefaultInputBufferSize = 512
	DefaultOutputBufferSize = 512
	DefaultBandwidth = 8 bytes/cycle

TABLE II: Details of the hardware blocks used In the experiments

main memory. Apart from the input/output and the bandwidth of the channels shown in Table II, we have kept all the other configurations as defaults in Multi2Sim [4]. After the workloads are simulated, the memory, network and processor execution reports are fed into McPAT [5]. McPAT estimates the power and area from given execution reports for a particular topology. Generated results are stored in a database. Using these results, we generate additional metrics such as area-delay product, computation per watt, energy consumed per area and energy - delay product.

B. *Nomenclature of Generated Topologies*

The nomenclature we have used is presented in Table III. We define the number of core with *C*. When data and instruction caches are shared, we represent these caches as *L* concatenated with the level it belongs. When instruction and data caches are separate, they are specified as *IL* and and *DL* respectively. If there exists a bypass connection, it is shown with *BP*. Thus, *2C_4L1_1L2* explains that there exists two cores, four L1 shared caches and one L2 shared cache. The topology *2C_2DL1_2IL1_1DL2_BP* shows us that there exists two cores with two data L1 cache, two instruction L1 caches, one data L2 cache and there exists a bypass connection from instruction caches to the memory. This nomenclature should be followed to comprehend our evaluations which are explained in the next section.

Symbol	**Definition**
C	Core Count
L<x>	Shared Data/Instruction Level 1, 2 or 3 Cache
IL<x>	Instruction Level 1 or 2 Cache
DL<x>	Data Level 1 or 2 Cache
BP	Exists a bypass connection

TABLE III: Nomenclature of the Topology Representation

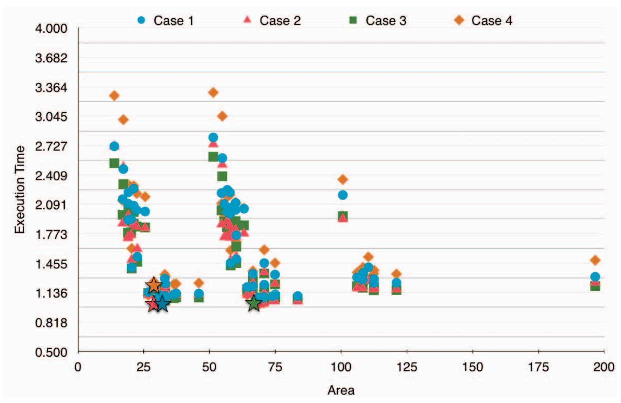

Fig. 4: Comparison of Execution Time vs Area

C. *Evaluations from Simulation Results*

In this section, we present our evaluations and emphasize two aspects. At first, we will summarize the best topologies in area, performance, energy consumption, energy-delay, area-delay, computation per watt metrics and area. In addition, we will present several metrics based on our evaluations that we recommend to include while evaluating systems. Optimizing a system for just one workload could generate poor results for other workloads. This hurdle can be alleviated by analyzing topologies with respect to certain design metrics in order to understand their similarities. In addition, the designers could have a number of requirements and these requirements could be conflicting such as high performance and low power. Nevertheless, identifying the suitable topologies which meet design requirements requires considering multiple objectives [23].

Area: Among our topologies, the topology that has the smallest area is *2C_1L1_1L2*. The L1 and L2 caches are shared among each core for instruction and data caches. In terms of performance *2C_1L1_1L2* has only surpassed *2C_1L1_1L2_1L3*. Adding a shared L3 cache has lowered the performance compared to *2C_1L1_1L2* for all cases. In terms of power consumption and leakage power, *2C_1L1_1L2* is the best topology. The main reason for low power figures is the low capacitance due to having the smallest chip area among our topologies.

Execution Time: In terms of execution time, topologies with four cores dominate our results. For cases 1 and 2, *4C_4DL1_1IL1_1DL2_1IL2* which is a hybrid topology performs the best. For cases 3 and 4, *4C_4DL1_1IL1_1DL2_1IL2_1L3* and *4C_8L1_1L2* have pro-

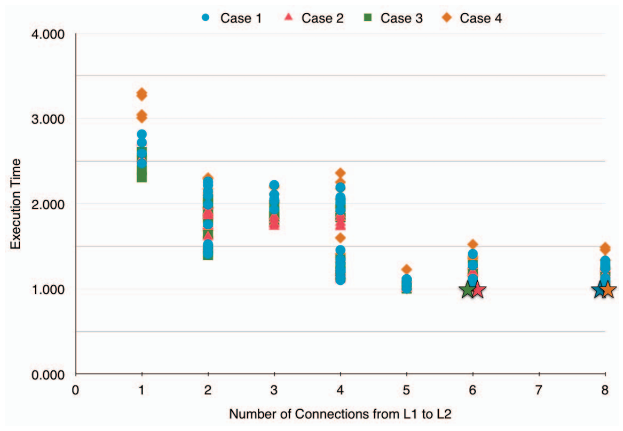

Fig. 5: Comparison of Execution Time vs Number of L1 to L2 connections

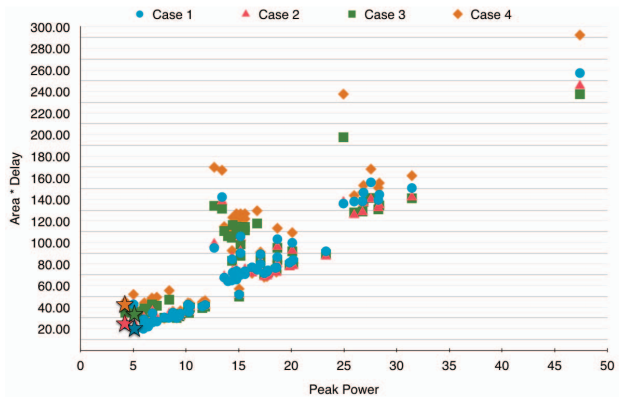

Fig. 6: Comparison of Area * Delay vs Peak Power

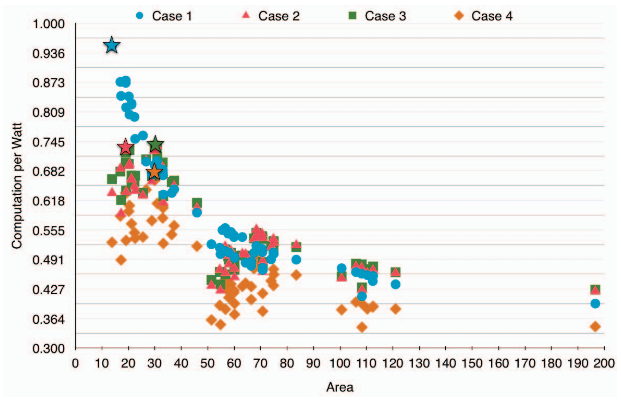

Fig. 7: Comparison of Computation per Watt (CPW) vs Area

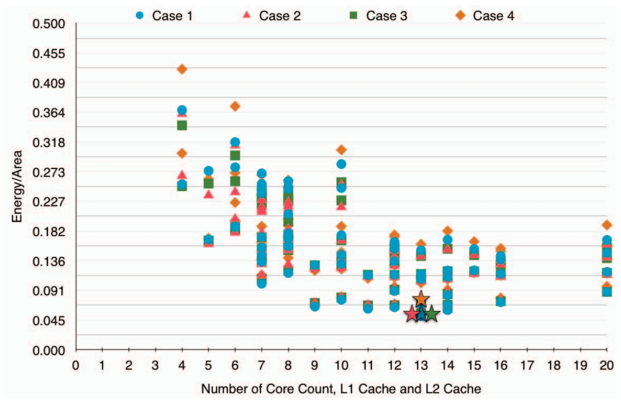

Fig. 8: Comparison of Energy per Unit Area (EPA) vs Number of Cores and L1 Cache

vided the best performance respectively. We have observed that the addition of L3 cache to *4C_4DL1_1IL1_1DL2_1IL2* topology has boosted the performance for case 3 which is the most compute intensive case among our workloads. In addition, it has surpassed all regular topologies in terms of performance. Figure 4 shows the execution times of all topologies for all cases and the best topologies are shown with stars. There are two groups of topologies which have lower execution times than the rest. When we observe these groups, we see that one group has smaller area than the other group. From our experimental data we have found out that there are hybrid topologies which can provide similar performance to regular topologies with L3 caches. Thus, exploring hybrid topologies have provided us to find out topologies that are smaller than regular topologies.

Based on our observation that the best execution time of different workloads could happen with different topologies, we have tried to identify if there are certain topologies which are more favorable than others. Figure 5 shows the execution times of different workload cases with respect to the Number of connections between L1 and L2 caches. There are two favorable topologies with 5 and 8 connections. Among these topologies *4C_8L1_1L2_1L3* and *4C_4DL1_1IL1_1DL2_1IL2_1L3* are the most favorable for all cases. The irregular *4C_4DL1_1IL1_1DL2_1IL2_1L3* topology has smaller area than the conventional *4C_8L1_1L2_1L3*, thus it is more preferable. In addition, although *4C_8L1_1L2_1L3* topology does not have the best execution time for a single case, it has shown that for case 1 and case 2, both cases can run on this topology relatively fast.

Area-Delay Product: Area-delay product is an important metric in order to determine the best architectures under area constraints. It allows designers to balance performance and chip-area which directly affects the cost of the system. Our experiments have shown that three topologies are favored in this metric. For case 1 and case 2, *4C_4DL1_1IL1_1DL2_1IL2* hybrid topology, for case 3 *4C_8L1_1L2* and for case 4 *4C_4L1_1L2* regular topologies have the lowest area-delay product. In terms of area, *4C_4DL1_1IL1_1DL2_1IL2* is smaller than *4C_8L1_1L2*. In case 3, the area-delay product of *4C_4DL1_1IL1_1DL2_1IL2* is very close to *4C_8L1_1L2*. Thus, it could also be selected as an alternative. For case 4, *4C_4L1_1L2* excels other topologies with its relatively small area and its four core topology decreases the execution time.

Unlike previous examples, certain design metrics could be

Workload	Execution Time	Power-Delay Product	CPW
Case 1	4C_4DL1_2IL1_2DL2_1IL2_1L3	4C_4DL1_2IL1_2DL2_1IL2_1L3	4C_4DL1_2IL1_2DL2_1IL2_1DL3_BP
Case 3	4C_4DL1_2IL1_2DL2_1IL2_1DL3_BP	4C_4DL1_2IL1_2DL2_1IL2_1DL3_BP	4C_4DL1_2IL1_2DL2_1IL2_1L3

TABLE IV: An example which shows three metrics for two cases and one metric is conflicting (CPW) with the other two

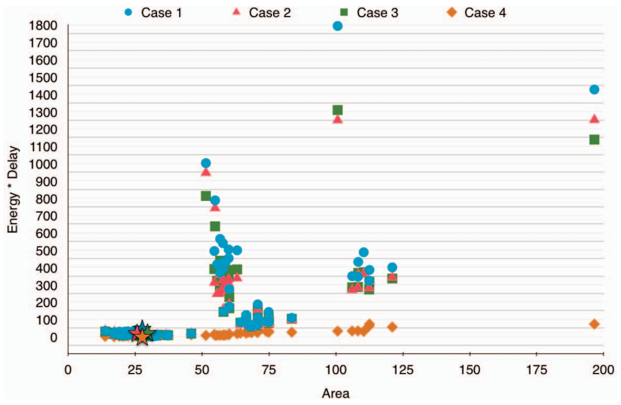

Fig. 9: Comparison of Energy * Delay (ED) vs Area

less dependent on the workload such as peak power. Peak Power is a crucial metric for electronic systems. Higher peak power increases the cost of the chip because it requires electronic components that are durable to peak power figures. Our simulations have shown that peak power is more dependent on the underlying topology. Nevertheless, metrics like peak power should not be omitted while designing architectures, because peak power is a decisive metric for low cost embedded systems. Thus, we have observed peak power in Figure 6 with Area-Delay product. For each case, we have seen that each workload favors a different type of topology for minimal peak power. In case 1, *2C_2DL1_2IL1_1DL2_BP* bypass topology has the lowest peak power. Our simulations have found out that, in case 2 and case 4, the regular topologies *2C_4L1_1L2* and *2C_1L1_1L2* have the lowest peak power. In case 3, the irregularity of *4C_4DL1_1IL1_1DL2_1IL2* topology helps to keep the peak power relatively low in this particular workload. The aforementioned architectures are shown with stars in Figure 6.

Computation per Watt: Computation per watt (CPW) metric is used in embedded systems and measures how the consumed energy is efficiently used for the actual computation. The calculation for the computation per watt metric is the execution time divided by the runtime power. Our experiments have shown that for different cases, architectures with highest CPW have different chip area, number of caches and L1 cache sizes. In more detail, for case 1, the topology with highest Computation per Watt is *2C_4L1_1L2*. However, for case 2 and 4, the bypass *2C_2DL1_2IL1_1DL2_BP* topology has the highest CPW. In case 3, the hybrid *4C_4DL1_1IL1_1DL2_1IL2* topology has the highest CPW. The selected points are shown with stars in Figure 7.

Power-Delay Product: When designing low power processor, power-delay product metric is used a lot because it allows designers to compare performance and power consumption of different topologies. In this metric, bypass topologies provided lower power delay products than other topologies. For case 1, case 2 and case 4, *2C_2DL1_2IL1_1DL2_BP* has the lowest power delay value. For only case 3, *2C_4L1_4L2* which is regular topology has dominated the other topologies. When we investigate the *2C_2DL1_2IL1_1DL2_BP*, we observe that its chip area is within 0.2% of *2C_4L1_1L2* which is a regular topology. However, bypassing the instruction caches from level 2 to memory has provided additional performance. Thus, this has lowered its power-delay product.

Energy-Delay Product: Embedded systems from mobile phones to coffee machines have similar behaviors, in which they are idle most of the time and gets under load in short bursts. For instance, a mobile phone wakes up from an idle state when the user wants to stream a video. Similarly, a coffee machine is only active when it prepares coffee, then it switches to a state where it waits for user input and then consumes it. Therefore, the active state is captured by the runtime dynamic power, the idle states can be captured by the leakage power metric. We assume 70% idle and 30% runtime case and prepare the metric as their combinations. Energy-Delay product is an important metric, which combines energy consumptions of topologies at the same level of performance. In order to reduce the energy delay product, either performance of the processor must increase or consumed energy per instruction must be lowered. However, in general, low power designs generally allow to sacrifice performance in favor of energy. In Figure 9, we have shown that for each workload case, there is a group of architectures that have lower EDP for all workload cases. Thus among those suitable architectures, *4C_8L1_1L2* and *4C_4DL1_1IL1_1DL2_1IL2* topologies are the most preferable. The favorable architectures are identified with stars in Figure 9.

Energy per Unit Area: Energy per Unit Area (EPA) with respect to number of cores, L1 and L2 caches in Figure 8 presents that there are several architectures which are less energy hungry than others. We see that between 12 and 14, the architectures for all cases have lower EPA values. Nevertheless, as in CPW, the best topologies are quite different. For example, case 1 and case 4 have the lowest EPA with *4C_8L1_1L2* which is a regular topology. *4C_4DL1_2IL1_2L2* and *4C_4IL1_4DL1_2DL2_BP* topologies have similar EPA figures for case 2 and case 3 respectively. However, three of the topologies belong to a different group of architectures. The topology *4C_8L1_1L2* is regular, however *4C_4DL1_2IL1_2L2* and *4C_4IL1_4DL1_2DL2_BP* are hybrid and bypass topologies respectively.

D. Design metrics with conflicting requirements

We present an example from our experimental data where optimizing on a single objective could be misleading. Table IV presents a comparison between two topologies. Topologies *4C_4DL1_2IL1_2DL2_1IL2_1L3* and *4C_4DL1_2IL1_2DL2_1IL2_1DL3_BP* occupy same chip area. We compare the Execution Time, Power - Delay Product and Computation per watt (CPW) metrics and present the topology that outperforms. *4C_4DL1_2IL1_2DL2_1IL2_1L3* has better execution time and Power - Delay Product in case 1 but *4C_4DL1_2IL1_2DL2_1IL2_1DL3_BP* has better CPW. For case 3 the situation is the opposite. *4C_4DL1_2IL1_2DL2_1IL2_1DL3_BP* has better execution time and Power-Delay Product but low CPW in case 3. This example yields two important results. Firstly, the workload has an important impact on the choice of the architecture, because what is best for one workload is not the best for the other workload. Secondly, if a designer optimizes his/her system for one metric, which might be the maximum CPW in this example, the selected topologies might not be the best in other metrics, which are the execution time and power delay product for this example. Hence, the solution to the design problem for implementing an embedded application with a multicore architecture lies in adopting a multi-objective design strategy for meeting stringent design requirements in the embedded systems domain.

VI. Discussion and Conclusion

As our results demonstrate after exploring various on-chip architectures, we have found out unconventional topologies that have outperformed the regular topologies. Similarly, designing for just a couple of metrics might not produce the best system under today's stringent design requirements. In particular, we make two important remarks regarding the preceding conclusions:

Shared instruction cache and bypassing cache levels: With this work, we aim to emphasize that, although current research has mostly dealt heavily optimizing the architectures of hardware blocks such as cache structures, the topologies of the systems have not been investigated thoroughly. We have studied and shown that unconventional hybrid topologies and bypass topologies can generate better designs than regular topologies. Hence, we expect that the improvements of the exploration of unconventional topologies can also be more striking when the design space gets larger by introducing more design parameters such as cache and processor structures.

Designing with numerous metrics: We have presented several metrics and shown the best architectures from our experimental results. We conclude that in order to meet today's stringent design requirements, designers should aim for ways to design with numerous metrics by the virtue of designing systems that can both provide high performance and energy efficiency.

VII. Acknowledgement

Funding from the Turkish Ministry of Development under the TAM Project, number 2007K120610 was received.

References

[1] C. Silvano *et al.*, "Multicube: Multi-objective design space exploration of multi-core architectures," in *VLSI 2010 Annual Symposium*. Springer, 2011, pp. 47–63.

[2] D. Tarjan, S. Thoziyoor, and N. Jouppi, "Cacti 4.0: An integrated cache timing, power and area model," *HP Laboratories, Palo Alto, CA*, 2006.

[3] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, 2011.

[4] R. Ubal *et al.*, "Multi2sim: a simulation framework for cpu-gpu computing," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*. ACM, 2012, pp. 335–344.

[5] S. Li *et al.*, "The mcpat framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing," *ACM (TACO)*, vol. 10, no. 1, 2013.

[6] M. Monchiero, R. Canal, and A. González, "Design space exploration for multicore architectures: A power/performance/thermal view," in *Proceedings of the 20th Annual International Conference on Supercomputing*, ser. ICS '06. New York, NY, USA: ACM, 2006, pp. 177–186.

[7] Z. Wang and A. Herkersdorf, "Software performance simulation strategies for high-level embedded system design," *Performance Evaluation*, vol. 67, no. 8, pp. 717–739, 2010.

[8] M. A. Laurenzano *et al.*, "Characterizing the performance-energy trade-off of small arm cores in hpc computation," in *Euro-Par 2014 Parallel Processing*. Springer, 2014, pp. 124–137.

[9] R. Cohen and T. Wang, "Intel embedded hardware platform," in *Android Application Development for the Intel Platform*. Apress, 2014, pp. 19–46.

[10] "Intel ark product information," http://ark.intel.com/, accessed: 2015-05-30.

[11] H. Agrou *et al.*, "A design approach for predictable and efficient multi-core processor for avionics," in *Digital Avionics Systems Conference (DASC), 2011 IEEE/AIAA 30th*, Oct 2011, pp. 7D3–1–7D3–11.

[12] G. Mariani *et al.*, "Design-space exploration and runtime resource management for multicores," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 2, p. 20, 2013.

[13] M. Monchiero, R. Canal, and A. González, "Design space exploration for multicore architectures: a power/performance/thermal view," in *Proceedings of the 20th annual international conference on Supercomputing*. ACM, 2006, pp. 177–186.

[14] Y. Zhang, N. Guan, and W. Yi, "Understanding the dynamic caches on intel processors: Methods and applications," in *Embedded and Ubiquitous Computing (EUC), 2014 12th IEEE International Conference on*. IEEE, 2014, pp. 58–64.

[15] O. Ozturk *et al.*, "Multi-level on-chip memory hierarchy design for embedded chip multiprocessors," in *Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on*, vol. 1, 2006, pp. 8 pp.–.

[16] I. Loi *et al.*, "Exploring multi-banked shared-l1 program cache on ultra-low power, tightly coupled processor clusters," in *Proceedings of the 12th ACM International Conference on Computing Frontiers*, ser. CF '15. New York, NY, USA: ACM, 2015, pp. 64:1–64:8.

[17] D. Bortolotti *et al.*, "Exploring instruction caching strategies for tightly-coupled shared-memory clusters," in *System on Chip (SoC), 2011 International Symposium on*, Oct 2011, pp. 34–41.

[18] "Arm embedded processors product information," http://www.arm.com/products/processors/, accessed: 2015-05-30.

[19] M. R. Guthaus, *et al.*, "Mibench: A free, commercially representative embedded benchmark suite," in *IEEE International Workshop on Workload Characterization, 2001*. IEEE, 2001, pp. 3–14.

[20] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 72–81.

[21] "Eembc, industry-standard benchmarks for embedded systems," http://www.eembc.org, accessed: 2015-06-15.

[22] P. Sweazey and A. J. Smith, "A class of compatible cache consistency protocols and their support by the ieee futurebus," in *ACM SIGARCH Computer Architecture News*, vol. 14, no. 2, 1986, pp. 414–423.

[23] J. Panerati and G. Beltrame, "A comparative evaluation of multi-objective exploration algorithms for high-level design," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 19, no. 2, pp. 15:1–15:22, Mar. 2014.